MFTP: Virtual TCP window scaling using multiple connections

David J. Iannucci
John Lekashman
NAS Systems Development Branch
NASA Ames Research Center
Moffett Field, California

January 1992

Abstract

This paper describes a solution to the problem caused by the TCP 64-kilobyte window limitation. This limitation restricts the throughput that TCP can get over network paths for which the product of bandwidth and end-to-end delay is greater than 64 kilobytes. The solution, which resides wholly at the application level, uses multiple TCP connections in parallel to increase the total effective window size. The role of the Research Internet Gateways (RIGs) in the testing of the solution is discussed.

1 Introduction

The idea for mftp grew out of a desire to improve throughput for file transfers by making better use of available bandwidth. Mftp is a file transfer utility based in large part upon the common UNIX program *ftp* (the "m" has no literal meaning itself; it is only meant to suggest the use of *multiple* connections). Ftp implements the Internet standard File Transfer Protocol (FTP). The essential difference between mftp and ftp is that mftp allows the user to specify an arbitrary number of data-carrying logical "connections," across which the data are evenly distributed, whereas ftp supports only one (per the FTP specification [1]). Why this makes a difference is an issue of some subtlety, which hopefully is made clear by what follows. The version of ftp which was used as the basis for the implementation of mftp came from the 4.3 BSD (Berkeley Software Distribution) UNIX system. It was a natural choice because of the authors' familiarity with the Berkeley network programming paradigm, and because it is free from licensing restrictions.

2 The TCP Protocol

Mftp is meant to be useful in environments where file transfers are being performed over networks which exhibit a (relatively) high product of bandwidth (maximum number of bits per second) times propagation delay (the time it takes for a bit to traverse the network from source to destination). The real culprit in such cases is the Internet Transmission Control Protocol (TCP), which defines and manages the aforementioned logical connections and ensures the reliable delivery (i.e. in order, and without loss or duplication) of data across TCP/IP networks. TCP implements what is known as a *sliding window protocol*. Such a protocol retains a buffer of data (the "window") which partitions the data space of the file into 3 disjoint sections. Ahead of the window is data which has yet to be sent. *Inside* the window is data which has been sent, but for which receipt has not yet been acknowledged by the receiver. Each byte of data sent by TCP has a unique sequence number, and must be buffered by the sender for possible retransmission until an acknowledgement of its receipt has been returned. Behind the window is data which has been sent *and* acknowledged, and therefore need no longer be saved by the sender. As data are sent and acknowledged, the window "slides" forward. Its size is not fixed, but may shrink and expand during a session, up to a fixed (usually per-host) maximum. This maximum size is bounded by 64 kilobytes. In other words, no TCP window may be set, or grow larger than 64 kilobytes. This upper bound is determined by the TCP protocol itself, which allocates only 16 bits in its packet header to hold the size of the window.

3 The Problem

The relationship between the window size and bandwidth-delay product (which we'll abbreviate BDP) is quite direct. The BDP gives the theoretical maximum number of bits (or bytes, if you wish to think of it that way) which may be in transit on the network between source and destination. In order to optimize network utilization and throughput, we wish to keep as much of the "pipe" filled as possible at any time. However, by definition, all of these bytes which are in transit must be held inside the sender's TCP window. It is when the BDP of a network path exceeds 64 Kb that we have a problem. In this case there is capacity in the network which is effectively being ignored by the protocol. From a more practical angle, if the sender can transmit into the network enough data to fill up its window *before* an acknowledgment comes back from the receiver (that is, the sender is forced to stop and wait for acknowledgment because its buffer is full), then time has been wasted in which bytes might have been sent. Ideally then, the window size for a TCP connection should be set no smaller than the BDP of the path followed by the connection. [1]

A prime example of a network path whose BDP exceeds the TCP 64 Kb limit is a satellite-based T1 circuit. The T1 offers roughly 1.5 Megabits/second of bandwidth, and the propagation delay, constrained by the speed of light, is on the order of half a second (quite a long time from a network point of view). The BDP for such a serial circuit is on the order of 100 Kb. Even in today's terrestrial T1 networks (BDP for a circuit that crosses the continental US is on the order of 15 Kb) time and bandwidth are being lost because most vendors set window sizes to 4 or 8 Kb by default. Perhaps a more compelling example is that of a T3 circuit (45 Mbit/sec) that stretches from the east to the west coast of the continental US, which has a BDP of roughly 450 Kb. This type of service has recently become a reality in the Internet, with gigabit per second wide-area networking attracting a great deal of interest. It is for these reasons that a solution to this problem is so important now.

[1] In current implementations, window sizes are "hardwired" into operating systems, so that they're the same for all TCP connections on that host. Although some systems allow appropriately programmed applications to set window sizes on a per-connection basis, there is currently no way to discover a path BDP dynamically, and there's no guarantee that it will not change in mid-session as the result of the path being rerouted.

4 Potential Solutions

Replacing TCP with another protocol is not currently an acceptable option for our purposes. Then, obviously, we must find a way to increase the window size available to TCP. A fairly straightforward approach that has been proposed in the Internet community and implemented by a few vendors is to add an *option* (a sort of trailer) to the TCP packet header which will store a multiplier for the window size field. This means that the window size given in the main part of the header will be multiplied by the value in the option field to find the *actual effective* window size [2,3]. This is usually referred to as the "window scale option," and has several disadvantages, all related to increased complexity.

First, it requires a change to the TCP specification [4], and therefore to a multitude of vendor implementations. Although this is not expected to cause a loss of interoperability, nevertheless it may be a long time before one can expect to take full advantage of the enhancement in largely heterogeneous network environments. Additionally, the enormous increase in possible window size causes a break-down of other aspects of protocol dynamics, specifically the sequence numbering scheme. The original decision to allow for 32 bits of sequence space was no doubt based on calculations which depended on an upper bound on window size of 64 kilobytes. Because the window scale option allows the window to grow much greater than 64 Kb, the 32-bit sequence space is rendered insufficient to ensure the uniqueness of sequence numbers among outstanding data segments. Thus, a new "pseudo-protocol" (based on timestamps) has been proposed to rectify this problem and restore the stability of TCP. Another negative side-effect of using a very large window is the potential necessity for equally large retransmissions. When a segment gets lost in transit, often much of the window needs to be retransmitted. Thus, another new "pseudo-protocol" has been proposed to allow selective acknowledgment, i.e., the ability for the receiver to inform the sender of out-of-order segments which were received, so that the sender need only retransmit the one or more segments that were actually lost. These things all add complexity not only to the protocol specifications, but certainly also to their implementations. Usually these implementations end up residing in operating system kernels, making operating system software harder to maintain, and problems harder to diagnose.

The approach that was chosen at NAS is to move the solution up above TCP to the level of application programs (specifically ftp). The mftp application opens (via the Berkeley sockets interface) multiple TCP connections for data transfer, and distributes the data equally among them. The primary reason for this choice was simplicity and ease of implementation. Keeping the modifications away from the operating system kernel and away from the TCP protocol itself allowed a very simple and flexible solution to be developed that was easily ported to many different operating system platforms. By its nature it avoids all of the difficulties to which the above-mentioned approach is subject. The disadvantage of this solution is that its benefits are not available to other programs which use the TCP transport service. However, FTP is the application that interested us, and given that it exists in "user space," the algorithm can be patched into other codes with relative ease.

5 Implementation Details

It should be emphasized that mftp runs as a single process which manages multiple connections. It does not fork off a child process for each connection, as do some other applications which have need of more than one connection.

Mftp increases the effective window size *of a file transfer* by using multiple instances of TCP (multiple windows) in parallel. Thus the new effective window size is n times the window size that the individual TCP connections are using, if there are n such connections in use. This scheme shifts part of what would otherwise be TCP's responsibility – the need for the pieces of the file to be put together in the right order at the remote end of the transfer – onto the application program. [2] Mftp incorporates the following algorithm to perform this function: the n connections are visited by the program in a round-robin fashion for writing data (on the sending side) or reading data (on the receiving side). If it is possible to do so, the program will read or write one block of data (normally equivalent to one packet) from/to each connection in turn. If it is not possible, then mftp will skip over that connection

[2]Note that this problem is not nearly as hard as the ordered reassembly function that TCP performs. The fact that each individual TCP connection returns an in-order data stream takes a great load off of mftp's reassembly algorithm.

and move on to the next one (not possible means, on the sending side, that buffers are full and can accept no more data at that time, and on the receiving side, that buffers are empty because no data has been received from the network since the last time this connection was read from). It should be easy to see that an arbitrary connection is responsible for the transmission of file blocks whose positions in the sequential file space differ by n. For example, connection number 2 will transfer blocks whose indices are 2, $2+n$, $2+2n$, $2+3n$, etc. Mftp keeps a record for each connection which includes its current position in the file space. Once a connection finishes reading/writing an entire block at some position, it increments this file pointer by n blocks. Thus the file regains its proper order on the remote side, even if connections are randomly skipped over in the round-robin process as a result of packet loss in the network.

6 Verification

The mftp concept was subjected to a thorough battery of tests in the NAS Research Internet Gateway (RIG) wide-area testbed. The goals of these experiments were:

1. To determine the effectiveness of using multiple transport connections per data transfer to utilize bandwidth resources which would otherwise be lost in a "long fat network" (high BDP) environment because of the window size limitation of TCP.
2. To compare the effectiveness of the above scheme (multiple transport connections with relatively small window sizes) to the use of a single connection with a relatively large window for achieving the same goal.
3. To determine the effectiveness of "type-of-service" routing in enhancing the utility of the scheme listed in 1. Specifically, to compare the performance obtained by sending acknowledgments via the same (high BDP) path as the data to that obtained by sending acknowledgments via a separate, comparatively low delay, out-of-band path.

The RIG testbed is pictured in Figure 1. The RIGs are experimental prototype IP routers built by Proteon, Inc. under a DARPA contract. They were intended to be used by DARPA as the basis for a high-performance Defense Research Internet, for networking research, as opposed to carrying production traffic. The NAS RIG testbed served a similar purpose [5]. A RIG was installed at each of four NASA centers: Ames (Moffett Field, CA), Langley (Hampton, VA), Lewis (Cleveland, OH), and Marshall (Huntsville, AL). They were joined in a circular topology, with each pair connected by two circuits in parallel: a T1 satellite pipe (the thicker line) and a 56 Kbps terrestrial line. At each NASA center were one or two Sun workstations on an ethernet (which is denoted by MPT, for Multi-port Transceiver). All workstations were running SunOS 4.0 or higher, including slow-start TCP. Each of the 3 RIGs that were remote from Ames had an out-of-band console access connection through the X.25-based NASA Packet Switching System (NPSS).

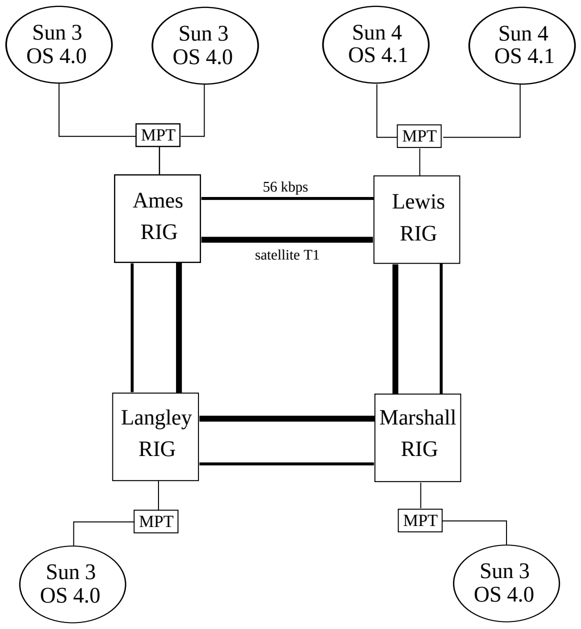


Figure 1

The basic paradigm for the experiments was a single file transfer between two endpoints separated by one or more serial “hops” through the RIG routers. All data-carrying packets, in all cases, passed over the satellite circuits. In the type-of-service-routed cases only, acknowledgement packets were returned via the low delay terrestial 56 Kbps circuit. Throughput measurements were taken directly from the out-put of mftp, which gives a short report at the end of each transfer. Further measurements were taken in the form of packet traces using a program called *tcpdump*. The traces allowed us to visualize the behavior of TCP by charting the number of packets sent and acknowledged as a function of time.

Experimental (input) variables and their values were:

- Number of connections: 1, 4, 8, 12, 14, 16, 18, 20, 23, 26, and 31
- TCP send/receive window size: 2, 4, 8, 16, 31, and 50 Kbytes
- End-to-end delay (measured in satellite "hops"): 1 and 2 hops, and 1 hop with type-of-service routing
- Transfer file size (fixed): 11.3 Mbytes

Two separate trials worth of data were collected for almost all possible combinations of the above variables. The ones omitted were ones which caused a problem which was not directly relevant to our testing activities (e.g., using 31 connections with 31 Kb windows caused the end hosts to freeze up due to exhaustion of network memory). For each stage of the experiment, the window size and delay variables were held constant. A shell script ran each stage mostly independent of human supervision (with the exception of host freeze-ups, router crashes, and the like). The script performed the following actions for each value of the connections variable:

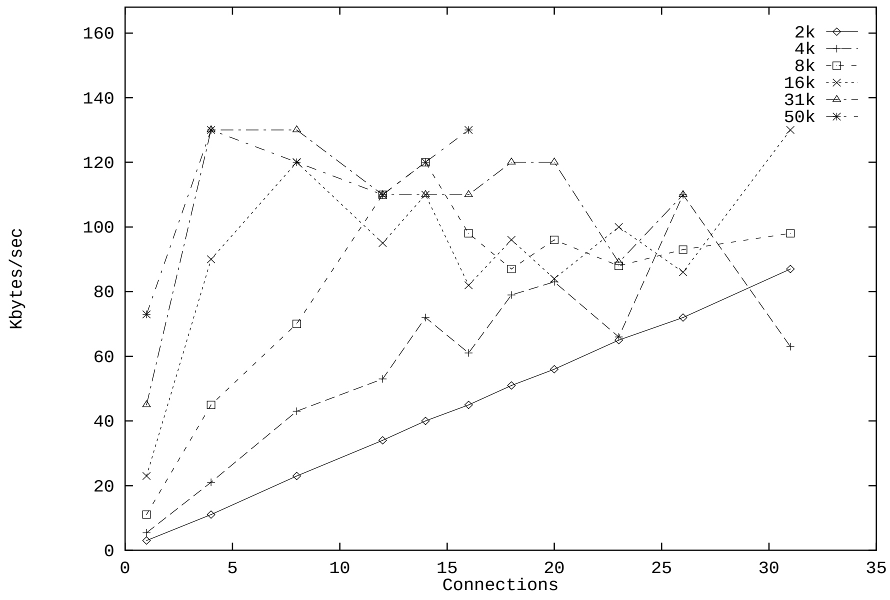
1. Spawn off a *tcpdump* process on the other local workstation. This process monitored TCP activity on that LAN.
2. Initiate an mftp session to the remote machine, and set the number of connections
3. Use the (m)ftp *put* command to transfer an 11.3 Mb text file into /dev/null
4. Save the result line from mftp, which reports the time and throughput for the transfer, into a file
5. Kill the *tcpdump* process.

7 Results

Our results showed the following:

1. Throughput increases almost linearly in the number of connections used as long as the total effective window size (TEWS) [3] is less than the bandwidth-delay product. This can be observed in the behavior of the graphs in Figure 2 for small numbers of connections and particularly for small window sizes. The erratic behavior exhibited at higher TEWS values will be explored below. Note that the graph slopes increase with window size, as would be expected (consider a slice of the graph taken at a fixed number of connections).

Figure 2: 1 satellite hop, per window size

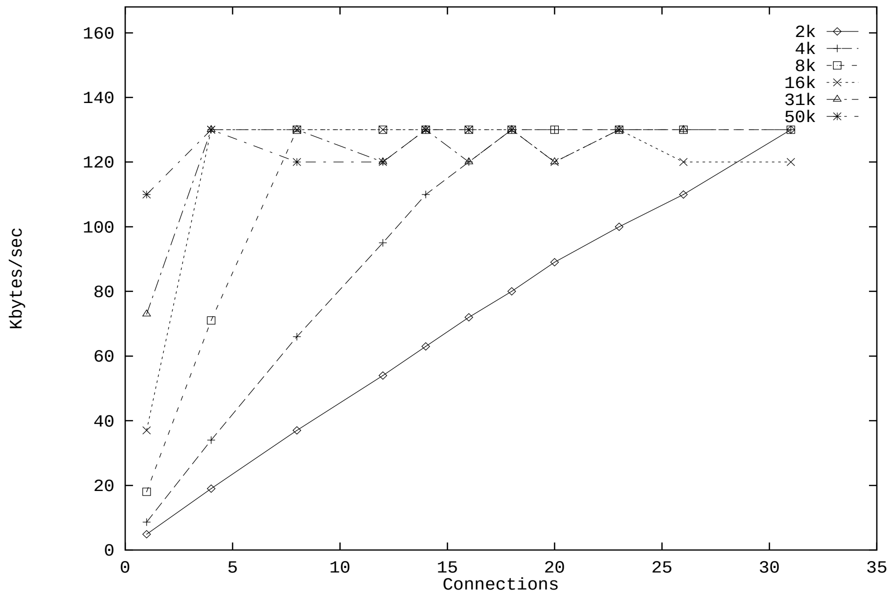


2. Throughput values are similar whether few large windows are used or many small ones, if the TEWS is the same. Compare the data points for 1 connection at 31 Kb, 4 connections at 8 Kb, 8 connections at 4 Kb, and 16 connections at 2 Kb in Figure 2.

[3]The configured individual window size times the number of windows (connections).

3. The use of type-of-service routing as in Goal 3 gives a significant throughput benefit over the "normal" case (compare the slopes of the graphs in Figure 2 to those in Figure 3). In this case, type-of-service routing also allowed us to completely fill the bandwidth of a single satellite T1 hop (which appears as the "ceiling" in Figure 3). The fact that we failed to do so in Figure 2 is a manifestation of the problem described below.

Figure 3: 1 hop, type-of-service routing, per window size



It is not at all clear exactly what was causing the erratic behavior. According to the *tcpdump* data for the one-hop case, [4] erratic effects begin to appear at well under the BDP. In fact, minor fluctuations occur even at very low TEWS values. It is notable that such behavior is virtually non-existent in all of the type-of-service routed cases (c.f. Figures 4 and 5). Graphically, the anomolies appear as tall spikes and deep valleys.

[4]In Figures 4, 5, 7 and 9, the "sent" graph shows the number of bytes transmitted or retransmitted since the last point on the graph. The "acked" graph shows the number of bytes *newly* acknowledged since the last point on the graph.

Figure 4: 18 conn, 16kb win, 1 hop (tcpdump)

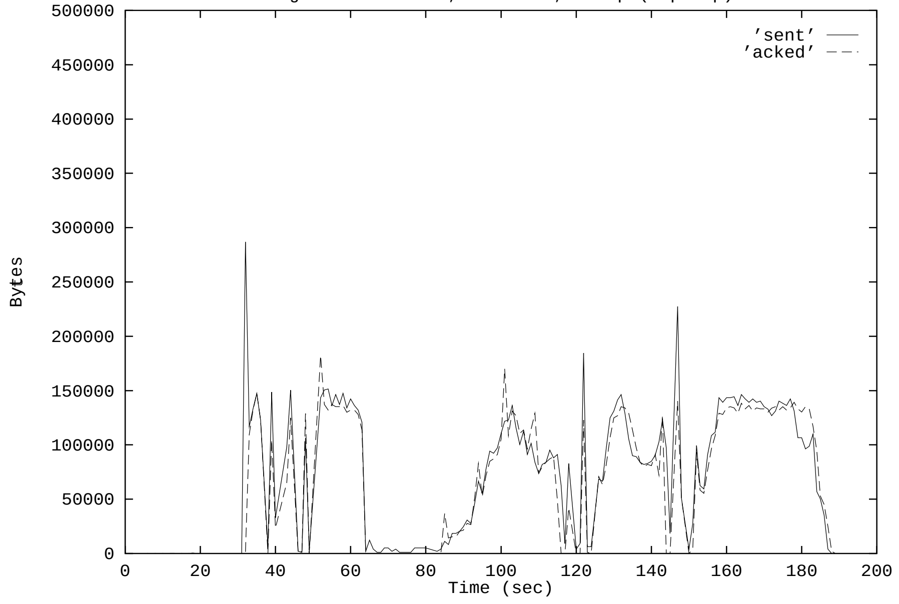
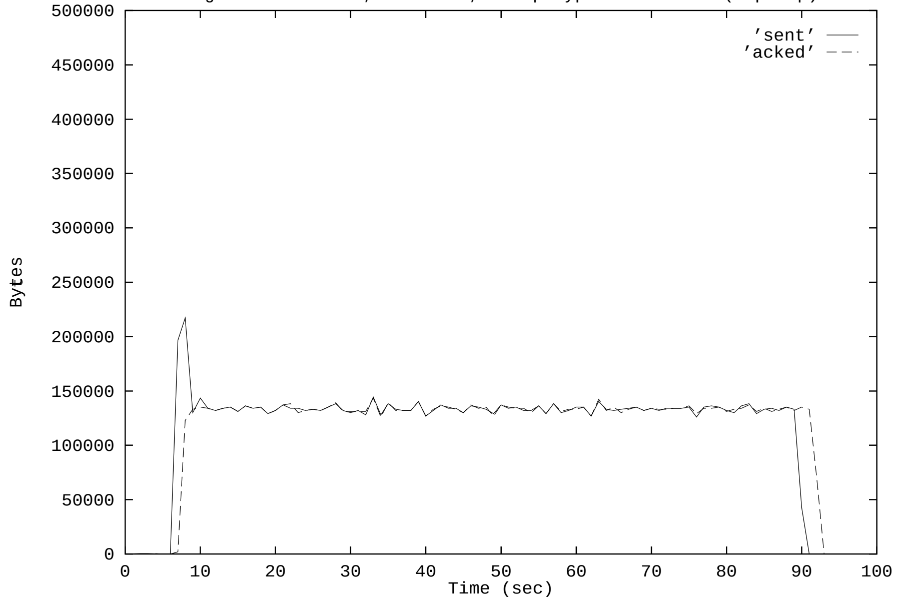


Figure 5: 18 conn, 16kb win, 1 hop type-of-service (tcpdump)



The most severe are valleys lasting on the order of 10-30 seconds, in which the total throughput drops to virtually zero. Close examination of these long blackouts reveals that they are caused by retransmission of a single packet sometimes 4 or 5 times before it is acknowledged. Congestion cannot be to blame because there are no (or very few) other packets flowing.[5] A most unusual aspect of this problem is that it seems not to be a "per-connection" phenomenon. When it happens, all, or almost all, of the connections are affected simultaneously. Furthermore, the problem appears to be router-interface-related. When ACKs were sent via a different interface, as in the type-of-service routed cases, these effects were totally absent.

Another unusual effect that we noted in the data concerns the results taken in the two-satellite-hop case. Looking strictly at the throughput numbers taken from mftp, there appears to be an artificially low ceiling governing the possible throughput over such a network. At first glance this may seem reasonable, since the end-to-end delay is twice that of the former cases. However, many of the data points in Figure 6 represent more than enough window space to cover the BDP of this network path (which is roughly 210 Kbytes). Therefore we should see numbers approaching the bandwidth of the network path (1.344 Mbps), as in the one-hop case. It is notable that the erratic effects seen in the one-hop case appear to be absent here, however if we look at the *tcpdump* graphs, we see that similar behavior is indeed there. It is not as random, nor as drastic, but it is more consistent (Figure 7 is a typical example). Most of the two-hop *tcpdump* graphs show two or three "blackouts" of the sort described above. Many also show throughput petering out significantly near the end of the transfer, sometimes reducing to less than 25% for the last full one third of the duration of the transfer.

[5]In further support of this, there appears to be no relationship between the intensity of erratic behavior and the number of ICMP source quench messages sent (usually zero).

Figure 6: 2 satellite hops, per window size

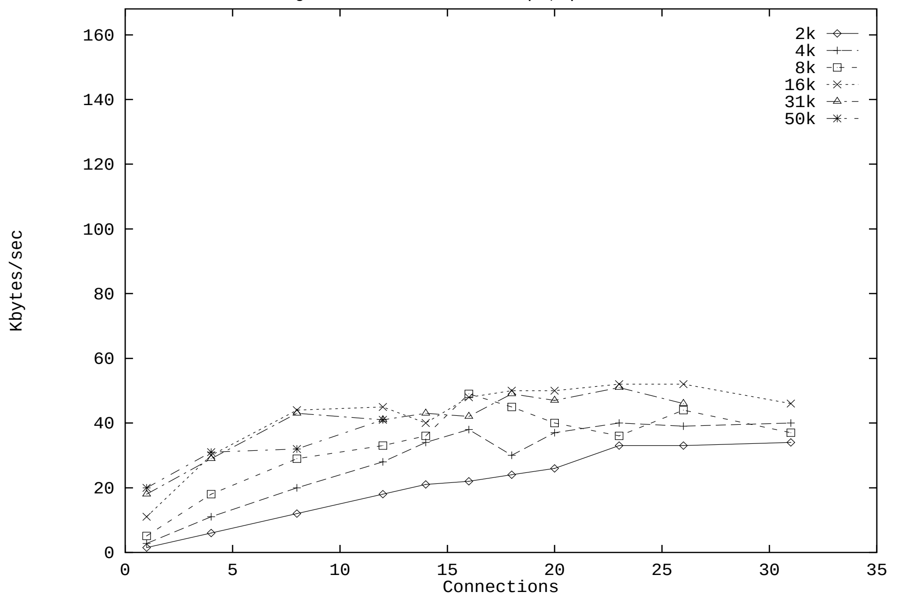
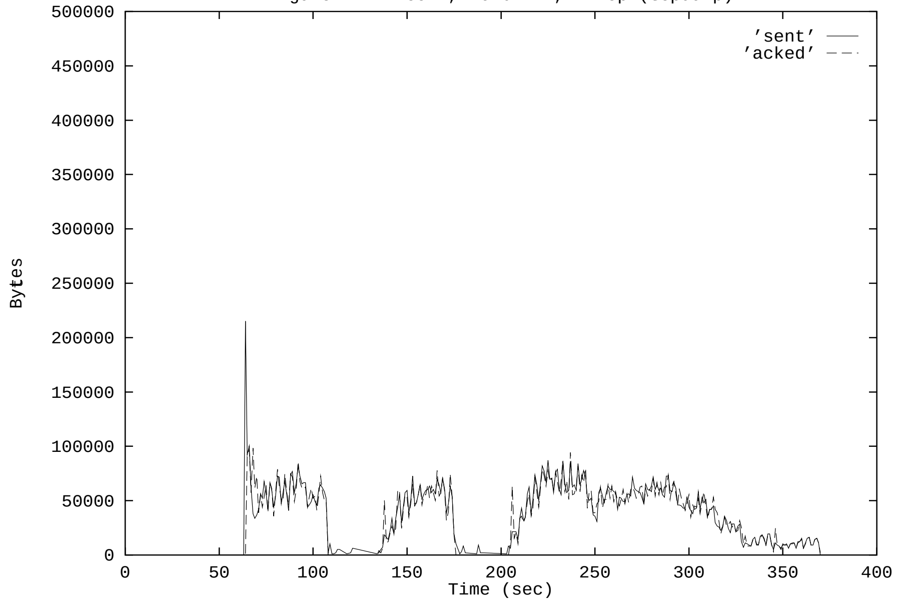
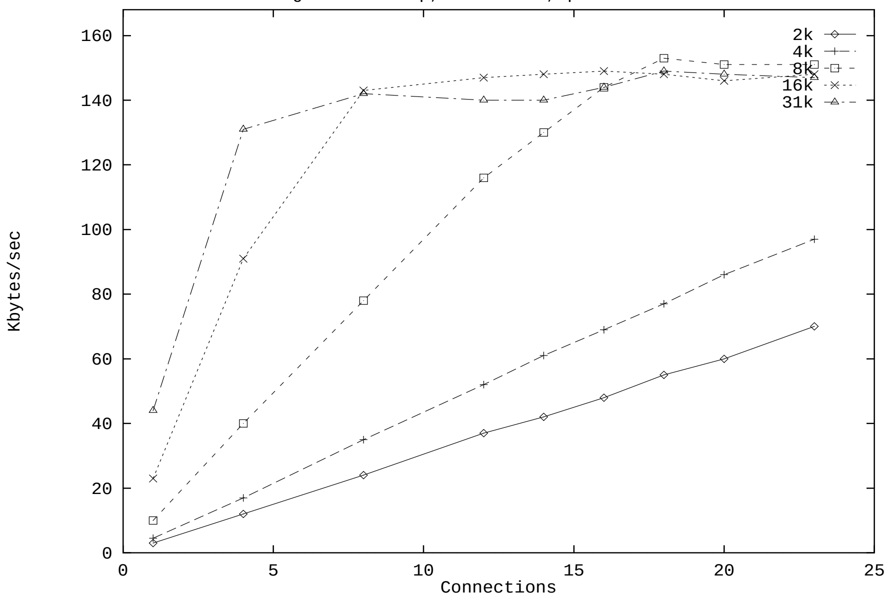


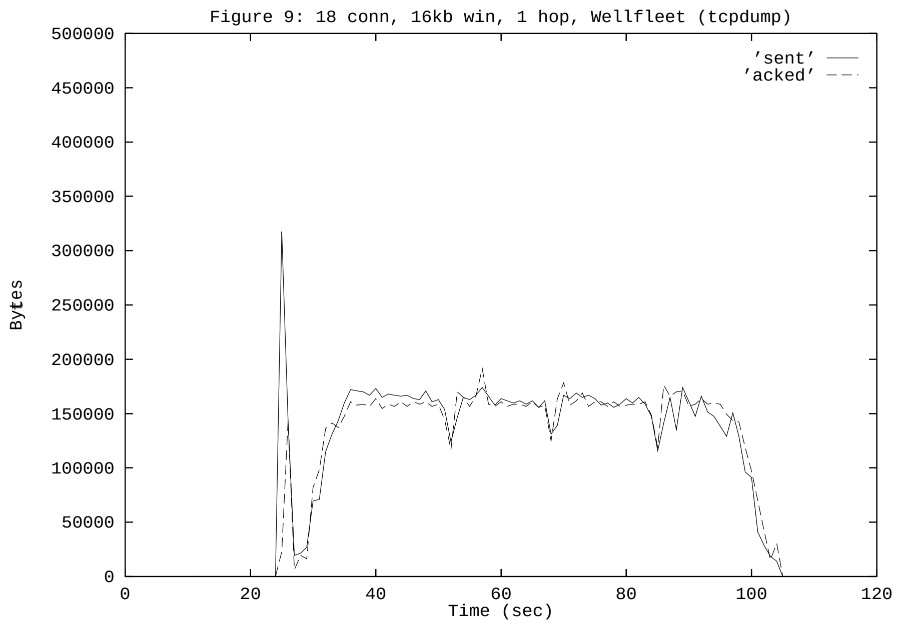
Figure 7: 14 conn, 16kb win, 2 hop (tcpdump)



After the wide-area testbed activities were completed, we performed additional "control" experiments in the Long Haul Communications Lab at the NAS so we would have some comparison data. The lab setup was the same as "half" of the wide-area testbed, i.e., two routers, each with an ethernet and hosts, connected by a serial line. The satellite T1 was replaced by a circuit simulator with a delay control capability. Two new data sets were taken from this testbed, corresponding to the use of RIGs vs. Wellfleet FN routers. The results reveal that the problem is being caused by a bug in the RIGs themselves. The RIGs performed the same in the lab as they had in the wide-area testbed, but the Wellfleet routers did not exhibit the erratic behavior. The graphs of the Wellfleet performance are shown in Figure 8 (c.f. Figure 2) and Figure 9 (c.f. Figure 4, and note the difference in time duration). The Wellfleet data also reconfirmed our previous observation that the system as a whole is less stable at higher TEWS values. Throughput numbers recorded for high TEWS values across multiple trials were usually different, while for low TEWS values they were often identical. This fact is reflected to some extent in the small perturbations at the top of Figure 8. The cause of this behavior could be the subject of further investigation.

Figure 8: 1 Hop, Wellfleet, per window size





8 Conclusion

The use of multiple TCP connections in parallel has been shown to be a successful solution to the problem of getting reasonable throughput over high bandwidth-delay product networks. Although it is not a truly general solution, it is highly portable (because it eschews any modification of protocols or operating systems) and will satisfy, in the short term, the needs of the majority of those TCP/IP users who are afflicted by the window limitation. Furthermore, type-of-service routing, as used in this case, has been shown to be an effective way to boost performance using a relatively small window. Less window space is needed because the decrease in round-trip delay made possible by sending acknowledgements via a low-delay path keeps the bandwidth-delay product small. [6].

[6]Note that, while the savings in host memory are scarcely reason enough to design a network this way, there are in fact other very good reasons for doing so (see [6]), and so one might as well take advantage of the situation.

9 References

1. Postel, J. and Reynolds, J., *File Transfer Protocol (FTP)*, RFC 959, USC ISI, October 1985.
2. Jacobson, V. and Braden, R., *TCP Extensions for Long Delay Paths*, RFC 1072, October 1988.
3. Jacobson, V., Braden, R. and Zhang, L., *TCP Extensions for High Speed Paths*, RFC 1185, October 1990.
4. Postel, J. ed., *Transmission Control Protocol*, RFC 793, USC ISI, September 1981.
5. Iannucci, D., *The Research Internet Gateways*, RND-92-001, NASA Ames Research Center, January 1992.
6. Lekashman, J., *Type of Service Wide Area Networking*, Proceedings of Supercomputing '89, Reno, Nevada, November 1989

Many thanks to David Yip and Jude George for last minute help with document prep, and to Michelle Koblas for reading and commenting.

For details on acquiring a copy of the mftp source code, please write to one of the addresses below:

John Lekashman
NASA Ames Research Center
MS 258-5
Moffett Field, CA 94035
Email: lekash@nas.nasa.gov